Title

A design framework for Internet-scale event observation and notification

Authors
Rosenblum, David S.
Wolf, Alexander L.

Publication Date
1997-02-03

Peer reviewed

A Design Framework for Internet-Scale Event Observation and Notification

David S. Rosenblum and Alexander L. Wolf

Technical Report 97-06
Department of Information and Computer Science
University of California, Irvine, CA 92697-3425

3rd February 1997

A Design Framework for Internet-Scale Event Observation and Notification

David S. Rosenblum
Dept. of Info. & Computer Science
University of California, Irvine
Irvine, CA 92697-3425
USA
dsr@ics.uci.edu
http://www.ics.uci.edu/~dsr/

Alexander L. Wolf
Dept. of Computer Science
University of Colorado
Boulder, CO 80309-0430
USA
alw@cs.colorado.edu
http://www.cs.colorado.edu/users/alw/

Abstract

There is increasing interest in having software systems execute and interoperate over the Internet. Execution and interoperation at this scale imply a degree of loose coupling and heterogeneity among the components from which such systems will be built. One common approach to designing distributed, loosely-coupled, heterogeneous software systems is a structure based on event generation, observation and notification. The technology to support this approach is well-developed for local area networks, but it is ill-suited to networks on the scale of the Internet. Hence, new technologies are needed to support the construction of large-scale, event-based software systems for the Internet. We have begun to design a new facility for event observation and notification that better serves the needs of Internet-scale applications. In this paper we present results from our first step in this design process, in which we defined a framework that captures many of the relevant design dimensions. Our framework comprises seven models—an *object model*, an *event model*, a *naming model*, an *observation model*, a *time model*, a *notification model*, and a *resource model*. The paper discusses each of these models in detail and illustrates them using an example involving an update to a Web page. The paper also evaluates three existing technologies with respect to the seven models.

Keywords: design, distributed systems, events, Internet, software engineering

1 Introduction

There is increasing interest in having software systems execute and interoperate over the Internet. Workflow systems for multi-national corporations, multi-site/multi-organization software development, and real-time investment analysis across world financial markets are just a few of the many applications that lend themselves to deployment on an Internet scale. Execution and interoperation at this scale imply a high degree of loose coupling and heterogeneity among the components from which such systems will be built. One common architectural style for distributed, loosely-coupled, heterogeneous software systems is a structure based on event generation, observation and notification. The technology to support this architectural style is well-developed for local area networks (e.g., Field's Msg [30], SoftBench's BMS [13], ToolTalk [16] and Yeast [19]), but it is ill-suited to networks on the scale of the Internet. Hence, new technologies are needed to support the construction of large-scale, event-based software systems for the Internet.

We envision event observation and notification as being an explicit facility provided to software components across the Internet. The facility would have the ability to observe the occurrence of events in components, to recognize patterns among such events, and to notify other, interested components about the (patterns of) event occurrences. This is a fairly simple and intuitive characterization of its requirements. However, this simple characterization masks the richness and complexity of the issues that must be addressed in the design and implementation of the facility. For example,

- To what extent should the facility support recognition of patterns of non-causally related events?
- What architecture will allow the facility to efficiently organize and partition its observation task, to handle notifications to multiple components interested in the same events, and to characterize events involving multiple components?
- Where in the architecture should the facility support event-pattern recognition and event information filtering?

These and many other questions must be carefully addressed in any design and implementation effort.

Recently there have been a small number of proposals and initial prototypes for Internet-scale event facilities, such as the OMG CORBA Event Service [26,27] and the TINA Notification Service [34]. But the definitions of these facilities address only a limited portion of the full problem space. Therefore, we have begun to design a new facility for event observation and notification that better serves the needs of Internet-scale applications.

In this paper we present results from our first step in this design process, in which we defined a framework that captures many of the relevant design dimensions. Our framework comprises seven models:

1. an *object model*, which characterizes the components that generate events and the components that receive notifications about events;
2. an *event model*, which provides a precise characterization of the phenomenon of an event;
3. a *naming model*, which defines how components refer to other components and the events generated by other components, for the purpose of expressing interest in event notifications;
4. an *observation model*, which defines the mechanisms by which event occurrences are observed and related;
5. a *time model*, which concerns the temporal and causal relationships between events and notifications;
6. a *notification model*, which defines the mechanisms that components use to express interest in and receive notifications; and
7. a *resource model*, which defines where in the Internet the observation and notification computations are located, and how resources for the computations are allocated and accounted.

Each of these models has a number of possible realizations. Taken together, these realizations define a seven-dimensional design space for Internet-scale event observation and notification facilities. Of course, these dimensions are not completely independent, because the models are interrelated in various ways. Because of these interrelationships, only a proper subset of the points in this space will correspond to adequate designs for Internet-scale facilities.

We describe these models fully in Section 3, but first in Section 2 we define more precisely what we mean by the notion of "Internet scale". In Section 4 we evaluate three existing technologies with respect to the design framework, and we conclude in Section 5 with a discussion of our plans for future work.

2 Attributes of Internet Scale

In order to provide an adequate design framework for an Internet-scale event observation and notification facility, we must first fully explore the ramifications of Internet scale. The primary distinguishing characteristics of an Internet-scale computer network are the vast numbers of computers in the network and the vast numbers of users of these computers. As a consequence of this, it would be infeasible to employ many kinds of low-level mechanisms that are used to support event observation and notification in a local-area network, such as the following:

- *caching and history mechanisms*, which retain persistent information about event occurrences in the network;
- *broadcast mechanisms*, which indiscriminately communicate event occurrences and notifications to all machines on a local network; and
- *vector clocks*, which piggyback onto each message exchanged between the communicating processes of an application a vector timestamp (whose size is linear in the total number of processes in the application), in order to aid the identification of causally-related events.

There are other characteristics of Internet scale that we can identify, and they are consequences of the vast numbers of participants.

One important related characteristic is the worldwide *geographical dispersion* of the computers and their users. As a consequence of geographical dispersion, it becomes necessary to address relativistic issues in multiple observations of the same event. For instance, observers of two events occurring on opposite sides of the world may observer two different orders for those events. Additionally, an application requesting a notification about an event at roughly the same time, but prior to, the occurrence of the event of interest may or may not be notified about the event.

At the scale of the Internet, the huge numbers of geographically-dispersed computers and users also have a much greater degree of *autonomy* than in local-area networks. Because of this autonomy, issues of resource usage are of greater concern, such as accounting and monetary charges for resource usage for observation and notification computations, what kinds of limits are to be placed on resource usage, and means of preventing misuse of resources or intrusiveness on others' usage of the resources.

Related to the issue of autonomy is the *security* of the computers and users. Mechanisms and policies must be established that will allow Internet-scale event observation and notification to take place in a manner that is compatible with security mechanisms such as firewalls, and is consistent with the need to enforce access permissions and other protection mechanisms.

Finally, concerns related to *quality of service* obtain much greater visibility at the scale of the Internet. Because of network latencies, outages and other dynamically-varying network phenomena, an Internet-scale event observation and notification facility will have to cope with decreased reliability of observations and notifications, as well as decreased stability of the entities to be observed and notified.

3 Design Framework

In this section we present a design framework for an Internet-scale event observation and notification facility. The framework is organized around the seven models listed in the introduction, each of which focuses on a different domain of concern in the design. Although the framework is general (in the sense of being independent of any particular application domain), we impose certain constraints that we feel are required in order for the facility to support true Internet-scale event observation and notification. And although the framework is quite comprehensive, there are aspects that it does not yet fully address, including a model of security and a model of quality-of-service; these are subjects of future work. Note that because the seven models are interrelated, it is necessary to defer the definitions of some concepts until the sections in which their relevant models are given full treatment.

Implicit in the relationships among the seven models is a timeline of activities involved in event observation and notification. We can identify seven such activities, which occur in sequence:[1]

1. *expression of interest* in an event or pattern of events;
2. *occurrence* of each event;
3. *observation* of each event that occurred;
4. *relation* of the observation to other observations to recognize the event pattern of interest;
5. *notification of* an application that its pattern of interest has occurred;
6. *receipt* of the notification by the application; and
7. *response* of the application to the notification.

We consider the last of these activities to be outside the domain of concern of the event observation and notification facility.

Looking at these activities from a slightly different perspective, our framework distinguishes three separate but related aspects of an event:

1. the *occurrence* of the event itself;
2. the *communication* of the fact of the occurrence to applications that are interested in the event; and
3. *information* about the event, some of which is general for all events (such as the time at which the event was observed), and some of which is specific to the event that occurred.

Two separate but related aspects of the communication include the *observation* of the occurrence and the *notification* of the occurrence. We consider notifications to be independent and unrelated. Any attempt by an application to relate in some way the different notifications it receives is a duplication of, and may be inconsistent with, the functionality of the event observation and notification facility.

3.1 Object Model

The object model for an Internet-scale event observation and notification facility incorporates the usual notion of encapsulation of functionality, which transcend considerations of Internet scale. An *object* can be a processor, storage device, network device, or some other hardware component of the network, as well as any logical entity

[1] The fact that there are seven models and seven activities is purely coincidental.

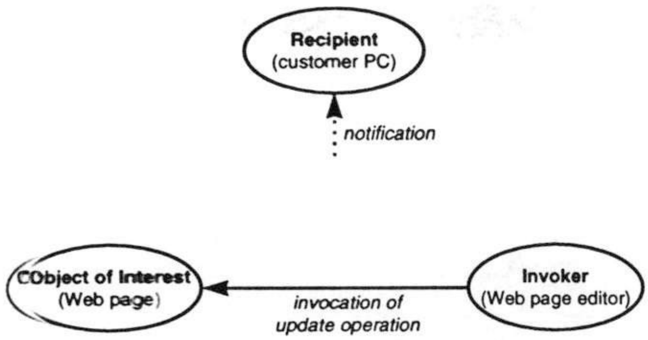

Figure 1. An Object Model for Web Page Updates.

residing on a hardware component, such as a file, a program, a process, a communication packet, and the like.[2] Humans also fit into this model in that we assume that they always have computer-based proxy objects working on their behalf. An object supports a set of *operations*, each of which can be *invoked* by some other object. We refer to an object whose operation is invoked as *an object of interest*, and we refer to the object invoking the operation as the *invoker*. An operation may be invoked directly through some apparatus associated with the object, or it may be invoked indirectly as a result of executing some program or software tool. Objects are also the entities that are recipients of notifications about events; we refer to such objects as *recipients*.

Figure 1 presents a simple example illustrating the concepts of the object model. The example involves three objects—a Web page object (the object of interest), an object that updates the Web page (the invoker), and an object that receives notifications about the update (the recipient). The operation applied to the object of interest in this case is an *update* operation, which replaces the contents of the Web page with new contents supplied by the invoker.

3.2 Event Model

The event model for an Internet-scale event observation and notification facility incorporates a straightforward notion of event. An *event* is the instantaneous effect of the (normal or abnormal) termination of an invocation of an operation on an object, and it *occurs* at the location of that object. An event can be uniquely characterized by the identity of the object of interest involved in the event, the identity of the operation that was invoked, the identity of the invoker, and the time of occurrence of the event.[3] An event is *observable* if it is possible for some object other than the object of interest and the invoker to detect the occurrence of the event. We refer to an observing object as an *observer*.

A consequence of this model of events is that there is a one-to-one correspondence between operation invocations and event occurrences. However, not every event will result in an observation of the event, and not every observation will result in a notification being communicated to some recipient. An event is simply a phenomenon that occurs regardless of whether or not it is observed. In other words, an event "costs" nothing; any costs that are incurred result from observations and notifications.

Looking again at the Web page example, Figure 2 depicts the event that is the effect of the termination of the update of the Web page. This event is observable, since a Web browser could be used to load the old version of the page prior to the occurrence of the event and the new version after the occurrence.

---

[2] While hardware objects and their operations may be of interest to applications such as network managers, in this paper we will concern ourselves solely with applications involving software objects and operations.

[3] The notion of identity is an aspect of the naming model, while the notion of the time of an event occurrence is an aspect of the time model, both of which are discussed below.

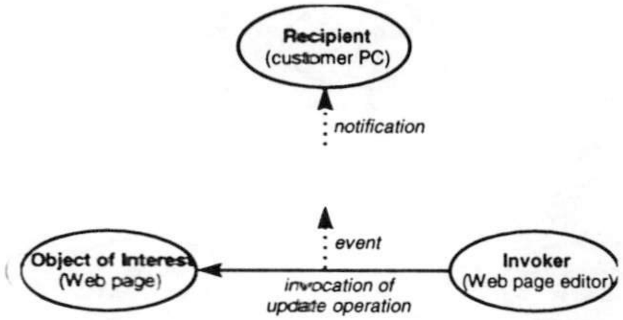

Figure 2. An Event Model for Web Page Updates.

3.3 Naming Model

Naming is of central importance in any software system [18], and this is especially true of the naming model for an Internet-scale event observation and notification facility, which provides a way of identifying events, as well as the objects, operations and other information associated with events. The naming model is employed for the purpose of expressing interest in events and requesting notifications about events. The realization of a naming model will typically offer a language that can be used to uniquely identify a specific event and to construct expressions whose interpretations are sets of events. In particular, the language will support the (possibly partial) specification of a name, for which there may be multiple matching event occurrences. We use the term *event kind* to refer to the set of event occurrences that can match a name.

The designer of an event observation and notification facility will have wide latitude in the choice of realization for the mechanism's naming model. The two most prevalent classes of naming model are *structure-based* and *property-based*. Structure-based naming models typically employ a hierarchical naming scheme that corresponds to the hierarchical organization of the entities of interest. The state-of-the-art in Internet-scale structure-based naming models is the Universal Resource Locator (URL), which provides a way of locating and accessing Internet resources [3]. URLs could be used as the realization for a mechanism's naming model, but the URL syntax and semantics would have to be extended to support the naming of additional kinds of objects; work in this direction is the subject of a draft specification for Uniform Resource Identifiers (URIs) [8].

In a property-based naming model, the entities to be named are named declaratively with a description of some property they possess or some predicate they satisfy. The current state-of-the-art in Internet-scale property-based naming models is to be found in Web search engines such as the AltaVista™ Search Service, which supports a content-based search mechanism for the location of Web pages.[4]

Figure 3 returns to the Web page example and depicts a possible syntax for naming the update event. A URL is used to identify the object of interest, while the standard hierarchical Internet domain naming scheme is used to identify the invoker. As was mentioned above, because this same name can be used to refer to all future instances of Web page updates by the invoker, we say that the update of the Web page by the invoker is a particular *kind* of event.

3.4 Observation Model

The observation model for an Internet-scale event observation and notification facility defines the way in which event occurrences and patterns of event occurrences are observed for the purpose of notifying interested recipients. Observation is achieved through a set of observer objects, and is implemented according to a number of policies that are defined as part of the model:

- an *observation policy*, which defines the mechanism by which observation of an event is achieved;

[4] AltaVista is a trademark of Digital Equipment Corporation.

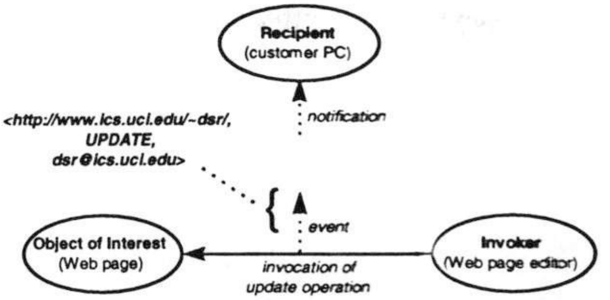

Figure 3. A Naming Model for Web Pages Updates.

- an *information policy*, which defines how event-specific information is to be requested and observed;
- a *pattern abstraction policy*, which defines what kinds of event patterns can be specified, how observer objects are configured to recognize event patterns, and how event patterns are to be identified for the purposes of requesting notifications about patterns;
- a *partitioning policy*, which defines the way in which observation tasks are partitioned among observers; and
- a *filter policy*, which defines how event-specific information is used to select events for notification.

There are other issues related to event observation that we discuss below as part of the resource model, such as when, where and how observers are created and destroyed.

As a consequence of Internet scale, it typically will be infeasible for the realization of the observation model to maintain histories of observations. Therefore, we expect most observation policies to preclude the persistence of observations. In other words, under such a policy, it would not be possible for a recipient to receive a notification about an event that occurred prior to the expression of interest in that event.

There are two classes of observation methods that can be employed for the observation policy: *synchronous observation,* in which the fact of an event occurrence is communicated explicitly to and in synchronization with the observer, and *polling*, in which the observer periodically checks for the occurrence of an event. Synchronous observation can be further subdivided according to whether the invoker communicates with the observer or whether the object of interest does. In all cases the observer eventually communicates a notification synchronously to one or more recipients and/or one or more observers.

Figure 4 depicts the Web page example with synchronous observations obtained from the invoker, while Figure 5 depicts synchronous observations from the object of interest. Figure 6 depicts an observer that uses polling to check for the Web page update event.

The information policy governs how event-specific information is requested, identified and observed. In particular, it must reconcile the desire of a recipient to request specific information about an event occurrence with the ability of an observer to obtain that information. For instance, in the case of the Web page update, a recipient may desire to obtain both the old contents of the Web page and the new contents of the Web page, to enable it to determine what was changed in the update. Thus, the recipient needs a way of expressing interest in both pieces of information so that the observer can take adequate steps to preserve the old contents prior to the occurrence of the update. In general, it will be unreasonable to support unrestrained requests for access to event-specific information, so the information policy must define precisely what kinds of request it can accommodate. One approach may be for recipients to provide the observer with a function or program that can be used to compute all desired information from the object of interest.

The pattern abstraction policy contains a definition of a language for specifying patterns of event kinds of interest. There are a number of suitable candidates for this pattern language, including general-purpose languages and logics such as regular expressions, first-order predicate calculus or temporal logic, as well as more specialized

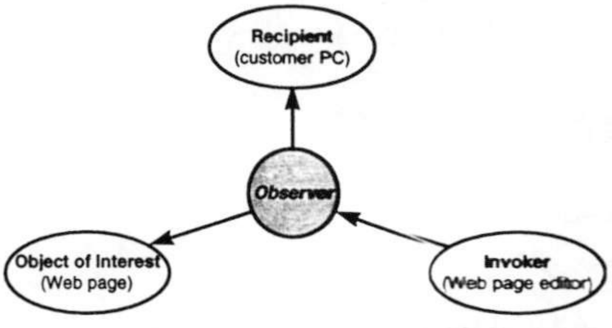

Figure 4. A Synchronous Observational Model for Web Page Updates (Synchronized with the Invoker).

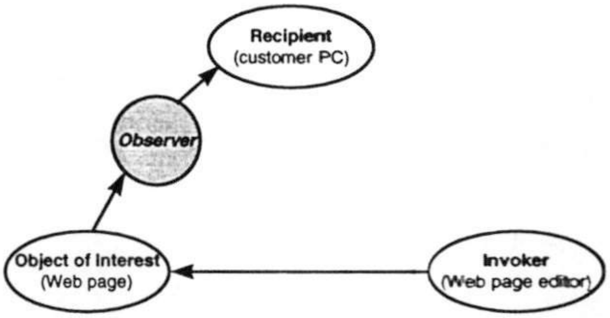

Figure 5. A Synchronous Observational Model for Web Page Updates (Synchronized with Object of Interest).

event-oriented languages such as TSL [22,31]. It is common to support *event abstraction* in order to provide a way of naming a pattern of events. Event abstraction is an especially notable feature of process algebras such as CCS [25]. The pattern abstraction policy may support a notion of event abstraction, in which a pattern of observed events is represented by a single abstract event or by a name that is used to refer to the pattern. Note that in order to treat a pattern as a true abstract event, it is necessary for the policy to establish some way of associating an object of interest, an operation and an invoker with the abstract event.

The issues associated with the partitioning policy can be framed using Table 1, which is a partial catalog of the possible cardinality relationships among events, observers, and recipients. For events, we can consider the cases of a single event, multiple independent events, or a pattern of events. For observers, we can consider a single observer or a team of observers working cooperatively. For recipients, we can consider either single or multiple recipients.

The first row represents the trivial situation of a single observer assigned to watch for a particular event and to notify a single recipient. The second row represents the situation in which multiple recipients share the services of a single observer watching for a particular kind of event; in essence, the event kind is associated with a dedicated observer. The third row, which says that there is a team of observers watching for the same event on behalf of a single recipient, makes sense in a context where fault tolerance is an issue and where redundancy can be used to alleviate the problem. Rather than going through all the remaining rows of Table 1, let us just point out some of the more representative situations that have been captured; the others can be inferred from the ones discussed here. The fifth row represents the situation in which a single observer is assigned the task of watching for multiple independent events on behalf of a single recipient; here, the observer has been dedicated to a recipient. The ninth row represents a situation similar to the first row, but in which there is a single observer watching for a pattern of events of interest to a single recipient. Instead of having a single observer watch for a pattern of events, we can have a team of observers watching for the pattern, as indicated by the eleventh row. This requires communication and coordination among the observers before a notification is sent to the recipient, but nonetheless it may be useful to distribute the observation task itself. Note that the table is only a partial catalog, in that one can conceive of further arrangements (such as, for

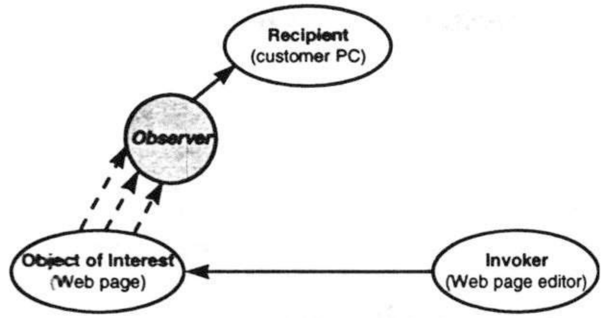

Figure 6. A Polling Observation Model for Web Page Updates.

	Events	Observers	Recipients
1	Single	Single	Single
2	Single	Single	Multiple
3	Single	Team	Single
4	Single	Team	Multiple
5	Multiple	Single	Single
6	Multiple	Single	Multiple
7	Multiple	Team	Single
8	Multiple	Team	Multiple
9	Pattern	Single	Single
10	Pattern	Single	Multiple
11	Pattern	Team	Single
12	Pattern	Team	Multiple

Table 1. Partial Catalog of Possible Cardinality Relationships Among Events, Observers and Recipients.

example, multiple recipients notified by a single observer watching for both multiple independent events and patterns of events), but space does not permit us to discuss the full range of possibilities.

In general, the choice among the various observation policies captured in Table 1 will come down to an issue of performance. Factors such as the rate at which events of a particular kind occur or the number and (physical or administrative) distance of recipients, must be understood before the "correct" policy can be chosen. Therefore, an event observation and notification facility should allow flexibility and dynamicity in how the observation task is partitioned.

Once a pattern of interesting events has been observed, a notification must be sent to the recipient. Whether that notification actually takes place depends on whether the information associated with the events can pass through any filter that has been established between the observer and the recipient. Notice that we are drawing here an important distinction between event *filters*, which are predicates on the content of associated information, and event *patterns*, which are predicates on the relationships among event occurrences. The filter policy is concerned with the language for expressing filter predicates, and where those predicates get evaluated, either at the observer or at the recipient. For instance, in the Web-page example, a recipient might be interested in only being notified of changes that involve more than 30% of the Web page. A predicate such as this highlights the fact that there is a general dependency between the associated information that is available and the filter predicate that can be expressed. In the example, the percentage of change must be somehow derivable from the information associated with the event.

3.5 Time Model

The problems of associating times with events in distributed systems and synchronizing clocks across distributed systems are well known (e.g., see Lamport [20]). But as a practical matter, the full ramifications of these issues are yet to be fully understood for networks of Internet scale. As we observe in Section 1, relativistic issues may preclude the use of any deterministic techniques for associating times and causal relationships with events. Internet-scale applications may therefore have to accommodate approximate representations of time, such as assuming the existence of a global clock even though such an assumption may result in inconsistent observations in different frames of reference.

Such issues are the concerns of the time model. An additional choice that must be made in the realization of the time model is the point or points at which times are to be associated with the activities involved in event observation and notification. With a synchronous observation model, either the invoker, the object of interest, or the observer of an event could have the responsibility of associating a time with the event. With a polling observation model, the observer would most naturally associate a time with the event; by necessity, this time would be approximate unless the time of occurrence can be derived from information about the event itself or the object of interest. For patterns of events, it may or may not be desirable to associate a time of occurrence; the time could be the time at which the first event was matched to the pattern or at which the last event was matched.

For the Web page example, the file system of the object of interest will associate a modification time with the new version of the Web page. It should be possible to use this modification time as the time of occurrence for the event.

3.6 Notification Model

The notification model for an Internet-scale event observation and notification facility is concerned with the communication between observers and interested recipients, which was illustrated for the Web page example in Figure 4, Figure 5 and Figure 6. In fact, this communication is bi-directional, since it involves, first, the expression of interest by a recipient in a particular pattern of events and, second, the communication of the notification along with any associated information that was requested.

Looking closer at the first direction of communication, we can see that there are essentially two ways in which it can be viewed. One way is to conceive of a pre-existing observer and a request being sent from the recipient to that observer. Another way is to treat the observer as the instantiation of an expression of interest.

Notifications themselves should be seen as independent communications between observers and recipients. This becomes particularly important when there are multiple independent observers involved. Attempts to relate notifications duplicate the job of the event observation mechanism, which is responsible for recognizing patterns of events.

A final issue related to notification is the lifetime of a recipient's expression of interest. The realization of the notification model must give a recipient the flexibility to specify whether it wants to be notified only upon the *first occurrence* of events matching its pattern of interest, upon *every occurrence*, or according to more complex characterizations such as *every Nth occurrence*.

Note that we could generalize our notification model somewhat to identify a separate *requester* or *broker* object, which establishes a relationship between an observer and a recipient. In other words, event observation and notification need not be initiated by the recipient. This model would accommodate applications that may be interested in forcing notifications to be sent to recipients, such as a software company wanting to notify customer PCs about product updates. The familiar *publish/subscribe* paradigm would be a degenerate case in which the object of interest and the observer together form the publisher, while the broker and recipient together form the subscriber.

3.7 Resource Model

An intriguing way to view an Internet-scale event observation and notification facility is that it is a particular architectural style for distributed computation in a wide-area network. Given that view, one can study the facility in terms of how resources in the network are allocated to carry out its computation. In our design framework this is the domain of the resource model.

The first consideration has to do with the specific architecture chosen within the style. The primary issue here is the computational independence of observers: are observation and notification simply part of the computation associated with invokers, objects of interest, or even recipients, or are they independent computations in their own right? A design that incorporates observation and notification with one of these other computations provides a straightforward answer to the question of which participant incurs the costs of observation and notification, but raises other questions, such as how to share observation and notification tasks. In contrast, if observers are independent computations, then there is greater potential for sharing. This independence, however, raises the question of where those computations take place and which participants are charged for those computations.

Related to the issue of architecture is the issue of managing the initiation and termination of the computations. Of course, invokers, objects of interest, and recipients all exist even in the absence of any event observation and notification tasks. So the resource model is specifically concerned with initiation and termination of observers. If observers are dependent computations, then clearly their lifetimes are tied to the objects within which they operate. If observers are independent computations, then a realization of the event observation and notification facility must provide some form of management mechanism.

3.8 Discussion

Considerations of Internet scale have influenced the formulation of each of our seven models to varying degrees. For instance, consider our event model, which may appear somewhat restrictive since its characterization in terms of an operation invocation implicitly limits an event to one invoker. Some events, such as meetings, may be more naturally characterized in terms of multiple invokers. But our formulation arises from Internet-scale considerations, since in general it would be infeasible to support the observation of an event involving multiple, Internet-wide invokers. Instead, events involving multiple invokers can be accommodated through event patterns in the observation model. Similarly, in the naming model, property-based naming may work well on an Internet-scale because it may be difficult or impossible to structurally name all events of interest. As we gain more experience with the design of our own facility, we expect to refine our models to incorporate additional constraints reflecting further considerations of Internet scale.

4 Evaluation of Existing Technologies

This section examines the space of existing technologies to determine the extent to which some of these technologies could serve as (the basis for) an Internet-scale event observation and notification facility, as well as to show how the design framework defined in Section 3 can be used to evaluate a candidate technology. A number of technologies are relevant to Internet-scale event observation and notification, and we can classify them as follows:

1. *theoretical models* of distributed clocks [20], vector timestamps [9,24] and partial orders of events [28];
2. *low-level event managers* for operating systems and windowing systems, such as the XView Notifier [15] and the Macintosh™ Toolbox Event Manager [6];[5]
3. the *implicit invocation* design model [10];
4. *languages and systems for event-based specification, analysis and debugging of software*, including Instant Replay [21], Event-Based Behavioral Abstraction [2], TSL [22,31] and Rapide [23];
5. *software buses*, such as Polylith [29], OLE/ActiveX [5] and CORBA [33];
6. *tool integration frameworks*, including Field [30], SoftBench™ [13] and ToolTalk™ [16];[6]

[5] Macintosh is a trademark of Apple Computer, Inc.

7. *communication and collaboration systems*, such as electronic mail, electronic bulletin boards, network news services [17], Lotus Notes®, and Corona [14];[7]

8. *software agent technology* (e.g., see Genesereth and Ketchpel [12]);

9. *active database systems*, such as AP5 [7] and Ode [11]; and

10. *event-action systems*, such as Yeast [19] and Amadeus [32].

Below we examine three particular technologies in detail—the Yeast Event-Action System, the CORBA Event Service, and the Network News Transfer Protocol. A more exhaustive evaluation of existing technologies will be the subject of future work.

4.1 Yeast

Yeast (Yet another Event-Action Specification Tool) is a client-server system in which distributed clients register *event-action specifications* with a centralized server, which performs event detection and specification management [19]. Each specification submitted by a client defines a pattern of events that is of interest to the client's application, plus an action that is to be executed in response to an occurrence of the event pattern. The Yeast server triggers the action of a specification once it has detected an occurrence of the associated event pattern. Higher-level applications are built as collections of Yeast specifications. These applications range from simple deadline notifications to comprehensive automation of activities in a software process.

Yeast's object model includes support for predefined object classes and user-defined object classes to Yeast. Yeast views an event as being a change to the value of an attribute of an object belonging to some object class. An event is named in Yeast's specification language by specifying the object class, object and attribute involved in the event, as well as an expression that the attribute must satisfy as an indication of the occurrence of the event. Yeast employs a hybrid observation model, using polling to identify occurrences of events involving predefined object classes, and a synchronous announcement mechanism to receive indications of occurrences of events involving user-defined object classes; the observations and specifications handled by one Yeast server are completely independent of those handled by any other Yeast server. For its time model, Yeast assumes the existence of a global clock, and it performs time zone conversions when the client and server are located in different time zones. Yeast's notification mechanism is the KornShell [4]. Communication from client to server is achieved through a number of Yeast client commands, while notification from server to client is achieved by executing the sequence of shell commands specified as the action of a specification. By default, any output produced by the commands of the action is sent by electronic mail to the user who submitted the specification. The Yeast server runs as a single UNIX® process and therefore has all of the computational privileges of the user that spawned the process.[8]

Because Yeast uses the TCP/IP protocol to implement all communication between client and server, it technically qualifies as an Internet-scale event observation and notification mechanism. However, the ability of a Yeast server to poll for events is limited to objects it can access in its local area network, typically via network file system services. Network transparency is also limited to a local area network, since at a minimum the client must specify the local network domain of the server with which it wishes to communicate. And although Yeast was designed as a general-purpose event-action system, existing implementations are suited primarily to observation of operating system-level events in networks of machines.

---

[6] SoftBench is a trademark of Hewlett-Packard Company. ToolTalk is a trademark of Sun Microsystems, Inc. See Barrett et al. for a recent study of event-based integration [1].

[7] Notes is a registered trademark of Lotus Development Corporation.

[8] UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/OPEN Company, Ltd.

4.2 The CORBA Event Service

The Common Object Request Broker Architecture (CORBA) is a general-purpose, Internet-scale software architecture for component-based construction of distributed systems using the object-oriented paradigm [26,33]. The CORBA specification includes specifications for a number of Common Object Services, one of which is the CORBA Event Service [27]. The CORBA Event Service defines a set of interfaces that provide a way for objects to synchronously communicate event messages to each other. The interfaces support a *pull* style of communication (in which the *consumer* requests event messages from the *supplier* of the message) and a *push* style of communication (in which the supplier initiates the communication). Additional interfaces define *channels*, which act as buffers and multicast distribution points between suppliers and consumers. The TINA Notification Service is a similar service defined on top of the CORBA Event Service [34].

The CORBA Event Service lacks support for many aspects of event observation and notification defined in Section 3. The object model is the object model of CORBA, and an event is simply a message that one object communicates to another object as a parameter of some interface method. The specification of the CORBA Event Service does not define the content of an event message, so objects must be pre-programmed with "knowledge" about the particular event message structure that is to be shared between communicating suppliers and consumers. Given this view of events, a naming mechanism is unnecessary, as is an observation mechanism, and any attempt to identify patterns of events is the responsibility of the consumers of event messages. Timestamps can be associated with events, but the meaning of such timestamps is at the discretion of the objects exchanging the event messages. Being a message, an event is its own notification. The computational aspects of events are subsumed by those of CORBA as a whole.

In summary, an event as defined by the CORBA Event Service really has no special semantics that distinguish it from any other method call in CORBA. We hope that future refinements of the CORBA specification will address more fully the phenomenon of event occurrences within CORBA applications.

4.3 The Network News Transfer Protocol

The Network News Transfer Protocol (NNTP) is the protocol used to distribute Usenet news articles (NetNews) across the Internet [17]. NetNews is organized into a collection of *newsgroups*, each one being set up to support ongoing discussion of a particular topic. Users express interest in a newsgroup by *subscribing* to it. A user can post an article to one or more newsgroups, whereby the article is distributed across a geographical reach specified by the user (although distribution of the articles posted to a newsgroup can be restricted according to policies established by the *administrator* of the newsgroup). As users post replies to articles they read, a *thread* is formed among a stream of related articles. At some point an article eventually expires.

One could reasonably view the newsgroups, the articles posted to the newsgroups, and the users who post the articles as being the objects recognized by NNTP. One could also view the reading of articles as being the key events, since responding to articles is the primary means by which new articles are generated. NNTP employs a simple hierarchical model for naming newsgroups, with articles numbered sequentially within a newsgroup and users identified by their electronic mail addresses. Except for the distribution specified at the time an article is posted, articles are broadcast indiscriminately across the Internet, making observation simply a matter of retaining unexpired articles of a newsgroup for any users who have subscribed to the newsgroup, and with threading being the sole pattern recognition task of the protocol. NetNews does not really require a notion of time except for the expiration of articles, and it suffices to assume the existence of a global clock for such a purpose. Users are notified about new articles by periodically running a news reading program, which will make available all new articles that have been posted to subscribed newsgroups. Computational limits are placed on the use of NetNews by system administrators, who may block access to or distribution of certain newsgroups and may establish expiration policies and storage limits for articles.

NNTP does an excellent job supporting an Internet-scale *publish/subscribe* model of communication. Several elements of NNTP do not quite correspond with our notion of event observation and notification.

5 Conclusion

We have described a design framework for an Internet-scale event observation and notification facility, to support construction of Internet-scale event-based distributed software applications. The framework comprises seven models that address seven different aspects of the design of the facility. We used this framework to evaluate three event observation and notification technologies representative of the state of the art.

We have several plans for future research on this problem. First, our design framework must better address security and quality-of-service issues, which could naturally be the subject of additional models in the framework. As we gain more experience in designing and constructing an Internet-scale event observation and notification facility, we will refine the models to incorporate lessons learned from our experience. A number of these refinements will likely be made to the observation and notification models, whose realizations will require careful engineering to ensure efficient and reliable operation on an Internet scale. Such refinements might involve the definition of a formal calculus of event operations that would support systematic optimization of the configuration of a network of observers, much in the same way that optimizations are applied to relational database queries in query languages such as SQL. Some operations that the calculus could support include generation, filtration, observation, notification, advertising, publication, subscription and reception.

Another key issue that must be addressed is the formal definition of the semantics of events that are of interest to applications. It is one thing to declare that a new kind of event that is to be observed, but in order to ensure that all occurrences of the event kind are generated uniformly, it will be necessary to provide a way of formally describing the semantics of the event kind and enforcing the semantics on objects to which they apply. Finally, it is clear that humans will play different roles in the use of an event observation and notification facility, but it is not yet clear how that role should be embodied in a user interface. The user interface will have to provide some scripting language or graphical means for the declaration of event kinds, the specification of event patterns of interest, and the generation of notifications. An important question to investigate, therefore, is whether the design of the user interface affects the design of all aspects of the facility itself, or whether it can instead be treated simply as just another application built on top of the facility.

Acknowledgments

The authors thank Rick Hall, Dennis Heimbigner, Alfonso Fuggetta, Antonio Carzaniga, André van der Hoek and Prem Devanbu for several fruitful discussions on the problem of Internet-scale event observation and notification.

References

[1] D.J. Barrett, L.A. Clarke, P.L. Tarr, and A.E. Wise, "A Framework for Event-Based Software Integration", *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 4, pp. 378–421, 1996.

[2] P.C. Bates and J.C. Wileden, "High-Level Debugging of Distributed Systems: The Behavioral Abstraction Approach", *Journal of Systems and Software*, vol. 3, no. 4, pp. 255–264, 1983.

[3] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)", Internet Engineering Task Force, Request for Comments RFC 1738, December 1994.

[4] M.I. Bolsky and D.G. Korn, *The New KornShell Command and Programming Language*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1995.

[5] K. Brockschmidt, *Inside OLE*. Redmond, WA: Microsoft Press, 1995.

[6] S. Chernicoff, *Macintosh*^{TM} *Revealed*, vol. 2: Programming with the Toolbox, 2nd ed. Indianapolis, IN: Hayden Books, 1987.

[7] D. Cohen, "Compiling Complex Database Transition Triggers", *Proceedings of SIGMOD '89: 1989 International Conference on Management of Data*, 1989.

[8] R. Daniel and M. Mealling, "Resolution of Uniform Resource Identifiers using the Domain Name System", Internet Engineering Task Force, Internet Draft (Work in Progress) 21 November 1996.

[9] C.J. Fidge, "Logical Time in Distributed Computing Systems", *IEEE Computer*, vol. 24, no. 8, pp. 28–33, 1991.

[10] D. Garlan and D. Notkin, "Formalizing Design Spaces: Implicit Invocation Mechanisms", *Proceedings of VDM '91: 4th International Symposium of VDM Europe on Formal Software Development Methods*, Noordwijkerhout, The Netherlands, 1991.

[11] N.H. Gehani and H.V. Jagadish, "Ode as an Active Database: Constraints and Triggers", *Proceedings of VLDB 91: 17th International Conference on Very Large Data Bases*, 1991.

[12] M.R. Genesereth and S.P. Ketchpel, "Software Agents", *Communications of the ACM*, vol. 37, no. 7, pp. 48–53 and 147, 1994.

[13] C. Gerety, "HP SoftBench: A New Generation of Software Development Tools", *Hewlett-Packard Journal*, vol. 41, no. 3, pp. 48–59, 1990.

[14] R.W. Hall, A. Mathur, F. Jahanian, A. Prakash, and C. Rassmussen, "Corona: A Communication Server for Scalable, Reliable Group Collaboration Systems", *Proceedings of CSCW '96 6th Conference on Computer Supported Cooperative Work*, Boston, MA, 1996.

[15] D. Heller, "The XView Notifier", *Unix World*, pp. 123–133, 1990.

[16] A.M. Julienne and B. Holtz, *ToolTalk and Open Protocols: Inter-Application Communication*: Prentice Hall, 1994.

[17] B. Kantor and P. Lapsley, "Network News Transfer Protocol", Internet Engineering Task Force, Request for Comments RFC 977, February 1986.

[18] A. Kaplan and J.C. Wileden, "Formalization and Application of a Unifying Model for Name Management", *Proceedings of ACM SIGSOFT '95 Third Symposium on the Foundations of Software Engineering*, Washington, DC, 1995.

[19] B. Krishnamurthy and D.S. Rosenblum, "Yeast: A General Purpose Event-Action System", *IEEE Transactions on Software Engineering*, vol. 21, no. 10, pp. 845–57, 1995.

[20] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System", *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[21] T.J. LeBlanc and J.M. Mellor-Crummey, "Debugging Parallel Programs with Instant Replay", *IEEE Transactions on Computers*, vol. C-36, no. 4, pp. 471–482, 1987.

[22] D.C. Luckham, D.P. Helmbold, D.L. Bryan, and M.A. Haberler, "Task Sequencing Language for Specifying Distributed Ada Systems (TSL-1)", *Proceedings of PARLE—The Conference on Parallel Architectures and Languages Europe, Volume II: Parallel Languages*, 1987.

[23] D.C. Luckham, J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification and Analysis of System Architecture Using Rapide", *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 336–355, 1995.

[24] F. Mattern, "Virtual Time and Global States of Distributed Systems", *Proceedings of Parallel and Distributed Algorithms*, 1988.

[25] R. Milner, *Communication and Concurrency*. Hemel Hempstead, Hertfordshire, UK: Prentice Hall International, 1989.

[26] Object Management Group, "The Common Object Request Broker: Architecture and Specification", revision 2.0, July 1995.

[27] Object Management Group, "Common Object Services Specification, Volume I", revision 1.0, March 1994.

[28] V. Pratt, "Modeling Concurrency with Partial Orders", *International Journal of Parallel Programming*, vol. 15, no. 1, pp. 33–71, 1986.

[29] J.M. Purtilo, "The POLYLITH Software Bus", *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 1, pp. 151–174, 1994.

[30] S.P. Reiss, "Connecting Tools Using Message Passing in the Field Environment", *IEEE Software*, vol. 7, no. 4, pp. 57–66, 1990.

[31] D.S. Rosenblum, "Specifying Concurrent Systems with TSL", *IEEE Software*, vol. 8, no. 3, pp. 52–61, 1991.

[32] R.W. Selby, A.A. Porter, D.C. Schmidt, and J. Berney, "Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development", *Proceedings of 13th International Conference on Software Engineering*, 1991.

[33] J. Siegel, *CORBA Fundamentals and Programming*. New York, NY: Wiley, 1996.

[34] Telecommunications Information Networking Architecture Consortium, "TINA Notification Service Description", July 1996.